# A Parabolic Load Balancing Method

Alan Heirich
Stephen Taylor

Computer Science Department
California Institute of Technology

Caltech-CS-TR-94-13

| Report Documentation Page | *Form Approved* *OMB No. 0704-0188* |
|---|---|

| | | |
|---|---|---|
| 1. REPORT DATE **2006** | 2. REPORT TYPE | 3. DATES COVERED **00-00-2006 to 00-00-2006** |

| | |
|---|---|
| 4. TITLE AND SUBTITLE **A Parabolic Load Balancing Method** | 5a. CONTRACT NUMBER |
| | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | 5d. PROJECT NUMBER |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Defense Advanced Research Projects Agency,3701 North Fairfax Drive,Arlington,VA,22203-1714** | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES
**The original document contains color images.**

14. ABSTRACT
**see report**

15. SUBJECT TERMS

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT **unclassified** | b. ABSTRACT **unclassified** | c. THIS PAGE **unclassified** | | **25** | |

# A Parabolic Load Balancing Method [1]

Alan Heirich & Stephen Taylor
*Scalable Concurrent Programming Laboratory*
*California Institute of Technology*

**Abstract**

This paper presents a diffusive load balancing method for scalable multicomputers. In contrast to other schemes which are provably correct the method scales to large numbers of processors with no increase in run time. In contrast to other schemes which are scalable the method is provably correct and the paper analyzes the rate of convergence. To control aggregate cpu idle time it can be useful to balance the load to specifiable accuracy. The method achieves arbitrary accuracy by proper consideration of numerical error and stability.

This paper presents the method, proves correctness, convergence and scalability, and simulates applications to generic problems in computational fluid dynamics (CFD). The applications reveal some useful properties. The method can preserve adjacency relationships among elements of an adapting computational domain. This makes it useful for partitioning unstructured computational grids in concurrent computations. The method can execute asynchronously to balance a subportion of a domain without affecting the rest of the domain.

Theory and experiment show the method is efficient on the scalable multicomputers of the present and coming years. The number of floating point operations required per processor to reduce a point disturbance by 90% is 168 on a system of 512 computers and 105 on a system of 1,000,000 computers. On a typical contemporary multicomputer [19] this requires 82.5 $\mu s$ of wall-clock time.

[1]The research described in this report is sponsored primarily by the Advanced Research Projects Agency, ARPA Order number 8176, and monitored by the Office of Naval Research under contract number N00014-91-J-1986.

## 1 Introduction

The scale of scientific applications and the computers on which they run is growing rapidly. Disciplines such as particle physics and computational fluid dynamics pose numerous problems which until recently have exceeded the memory and cpu capacities of existing computers. Moreover these disciplines appear ready to pose new problems which exceed the capacities of the largest computers that will be built in this decade. The *Grand Challenges* represent a set of problems solvable by computers with TeraFlops ($10^9$) performance. If present trends continue these will be supplanted within a decade by a set of challenges requiring PetaFlops ($10^{12}$) performance.

This increase in problem size is made possible by the dramatic reductions in size and cost of VLSI technology and parallel computing. Currently our research in computational fluid dynamics uses scalable multicomputers with roughly 500 processors [19, 23, 24]. Research efforts are underway to develop within three years time scalable multicomputers with hundreds of thousands of functional units [16, 21]. These trends suggest that limits to the growth of scientific computing are more likely to be determined in the next several years by software technology than by hardware limitations.

A potentially limiting software technology is the class of methods to solve the load balancing problem. Most numerical algorithms require frequent synchronization. If a load distribution on a multicomputer is uneven then some processors will sit idle while they wait for others to reach common synchronization points. The amount of potential work lost to idle time is proportional to the degree of imbalance that exists among the processor workloads. Since this loss also increases with processor count it can be valuable to control the accuracy of the resulting balance and to trade off the quality of the balance against the cost of rebalancing.

This paper presents a load balancing method for mesh connected scalable multicomputers with any number of processors. The method can balance the load to an arbitrary degree of accuracy. Theory and experiment show the method is inexpensive under realistic conditions. While it is effective on contemporary systems with under 1,000 processors it is specifically intended to scale to systems with tens and hundreds of thousands of processors. The total wall clock time for the method decreases as the processor count increases. The method was developed to solve the dynamic (run time) load balancing problem. It has also proven useful for cases of static (initial load time) balancing.

The method assumes the computation is sufficiently fine grained that work can be treated as a continuous quantity. This is reasonable in an application domain like computational fluid dynamics in which units of work represent grid points in a simulation. Each processor is typically responsible for thousands of grid points and can exchange individual points with other processors.

Cybenko [6] has published a method which is similar in several respects to this one. He proves asymptotic convergence of an iterative scheme on arbitrary interconnection topologies. Two other articles have appeared more recently on diffusive load balancing methods. Boillat [4] demonstrates polynomial convergence on arbitrary interconnection topologies using a Markov analysis. Horton objects to the polynomial convergence demonstrated in [4] and the lack of bounds on accuracy in the previous work. He applies a multigrid concept [11, 14] to accelerate convergence of a simple diffusive scheme.

The method presented in this paper, while developed independently, resembles a special case of Cybenko's method restricted to mesh connected topologies. It differs from Cybenko's method with regard to issues of numerical stability. The objections in [11] lack rigor. This paper refutes them via formal demonstrations of convergence rates, accuracy, and scaling. These results show that convergence is rapid, accuracy is limited only by machine precision, and superlinear speedup can be achieved for cases of practical interest in CFD.

## 2 A Parabolic Model

A number of articles have proposed solutions to the load balancing problem in recent years [2, 5, 10, 12, 13, 15, 17, 18, 22]. Many of these solutions are reliable and efficient for computer systems with small numbers of processors. Unfortunately many of them do not scale well to systems with large numbers of processors. It is well known that in a scalable algorithm the amount of work performed in parallel grows more rapidly than the amount of work performed in the serial part of the calculation as the size of the problem increases [1, 9]. Scalable algorithms tend to be highly concurrent and this fact often makes it difficult to prove that they are correct. A load balancing method for scalable multicomputers should be scalable but should not sacrifice reliability.

It is worth noting that a class of random placement methods have been proposed for scalable multicomputers [2, 10]. These methods are scalable and are reliable under the assumption that disturbances occur frequently and have short lifespans. These assumptions do not hold in a domain like CFD where disturbances arise occasionally and are long lasting. As a result we are unable to take advantage of the rewards these methods offer.

A method is *reliable* if it can be shown to compute a correct solution within a predictable number of steps. The simplest reliable load balancing method collects load statistics from all processors, computes the average load, and broadcasts the average to all processors. Each processor then exchanges work with it's neighbors so that the new workloads equal this average. Unfortunately this simplest method is not *scalable* because it is inherently serial. The number of terms in the calculation of the average load is proportional to the number of processors in the computer system. Although this cost can be made logarithmic with an octree data structure there is a more severe cost associated with interprocessor communication. The current state of the art in mesh routing technology requires a nonconflicting communication path for each message. The opportunities for path conflicts known as "blocking events" increase factorially with the number of processors in the computer system. This simplest reliable method is not scalable because the time lost to blocking events can grow factorially with the size of the computer system.

A method is *scalable* if it can run efficiently on computer systems with very large numbers of processors. Due to the effects of Amdahl's law [1] most scalable methods are concurrent algorithms in which the computation is distributed among the processors in the computer system. What these methods gain in scalability they often lose in reliability because they lack formal proofs of correctness and convergence.

As one example of the problems which can arise in concurrent algorithms consider a simple concurrent method in which each processor adjusts it's load to equal the average of the loads at it's immediate neighbors. This method is distributed and scalable and is easily seen to be convergent. Unfortunately it is well known that it converges to solutions of the Laplace equation $\nabla^2\Phi = 0$. This equation is known to admit sinusoidal solutions which are not equilibria. As a result this method, although scalable, is not reliable.

This paper leverages existing numerical and analytic techniques to derive a reliable and scalable load balancing method. The method is based on properties of the parabolic heat equation $u_t - \alpha\nabla^2 u = 0$. The heat equation describes a process in nature whereby thermal energy diffuses away from hot regions and into cold regions in a volume until the entire volume is of the same temperature. A literal interpretation of the terms in the heat equation reads that the rate of change in temperature $u$ at each point in the volume is determined by the local curvature $\nabla^2 u$ times a diffusion rate $\alpha$. This paper applies finite difference techniques to derive an unconditionally stable discrete form of this equation, and uses a scalable iterative method to invert the resulting coefficient matrix. The end result is a concurrent load balancing method which is scalable, reliable and efficient.

## 3 A Parabolic Algorithm

The algorithm consists of a simple arithmetic iteration which is performed concurrently by every processor in the multicomputer. Each step of the iteration requires 7 floating point operations at each processor. The iteration calculates an expected workload at each processor. Processors periodically exchange units of work with their immediate neighbors in order to make their actual workload equal to this expected workload. The algorithm contains parameters which control the accuracy of the resulting solution. In many applications an accuracy of 10% is sufficient. In these cases only 24 iterations are required to reduce a point disturbance by 90% regardless of the size of the multicomputer. This paper presents the algorithm for a three dimensional mesh. The reduction to two dimensions is described in the discussion section.

### 3.1 Initialization

Specify the accuracy $\alpha$ desired in the resulting equilibrium. For example, to balance to within 10% choose $\alpha = 0.1$. Determine the interval $\nu$ at which processors will exchange work with their immediate neighbors according to the formula

$$\nu = \left\lceil \frac{\ln \alpha}{\ln \frac{6\alpha}{1+6\alpha}} \right\rceil \geq 1 \qquad (1)$$

Note that in the interval $0 < \alpha < 1$ $\nu$ is less than or equal to 3:

$$\nu = \begin{cases} 2: & 0 < \alpha \leq 0.0445 \\ 3: & 0.0445 < \alpha < 0.622 \\ 2: & 0.622 \leq \alpha < 0.833 \\ 1: & 0.833 \leq \alpha \end{cases}$$

## 3.2 Execution

At every processor $x, y, z$ adjust the workload $u_{x,y,z}$ as follows:

```
$u^{(0)}_{x,y,z} = u_{x,y,z}$
for m=1 to $\nu$
    $u^{(m)}_{x,y,z} = \frac{u^{(0)}_{x,y,z}}{1+6\alpha} + \left(\frac{\alpha}{1+6\alpha}\right)\left(u^{(m-1)}_{x+1,y,z} + u^{(m-1)}_{x-1,y,z} + u^{(m-1)}_{x,y+1,z} + u^{(m-1)}_{x,y-1,z} + u^{(m-1)}_{x,y,z+1} + u^{(m-1)}_{x,y,z-1}\right)$    (2)
endfor
**Exchange** $\left(u^{(\nu)}_{x,y,z} - u'^{(\nu)}\right) \alpha$ units of work with every neighbor $u'$.
$u_{x,y,z} = u^{(\nu)}_{x,y,z}$
```

Repeat these steps until reaching equilibrium. Much of the following analysis will be concerned with formulating an exact statement of the number of repetitions required to reach equilibrium. The motivation behind this analysis is to support claims of correctness, convergence, and accuracy. The analysis develops a theory of convergence for any disturbance and applies this to the specific case of a point disturbance. Analysis appears to be less practical in many cases than conservative estimates derived from simulations. Following the analysis this paper presents simulations from cases of interest in CFD and distributed operating systems. The paper concludes with a summary and discussion.

## 4 Reliability and Scalability

This section demonstrates reliability of the method by showing that for any initial disturbance every component of the disturbance vanishes at an exponential rate. It demonstrates scalabilty by defining a lower bound on this rate as a function of $\alpha, n$. It applies the resulting theory to the case of a point disturbance and demonstrates that weakly superlinear speedup can be achieved under realistic assumptions. The appendix contains technical derivations in support of this analysis.

## ACCURACY OF THE JACOBI ITERATION

Since the algorithm is intended to observe strict accuracy of $O(\alpha)$ it is important to verify that each stage of the algorithm observes this accuracy. The justification of accuracy for the finite difference equation is given in the appendix. The accuracy of the coefficient matrix inversion can be verified by analyzing the spectral radius of the Jacobi iteration.

From the Geršgorin disc theorem [7] the eigenvalues $\lambda$ of (2) are bounded $|\lambda| \leq \frac{6\alpha}{1+6\alpha}$. Since the row and column sums are constant and the iteration matrix is nonnegative ([7], theorem 8.1.22) the spectral radius equals the row sum

$$\rho\left(D^{-1}T\right) = \frac{6\alpha}{1+6\alpha} \tag{3}$$

Define the error in a current value $\vec{u}^{(m)}$ under the iteration (2) as $e(\vec{u}^{(m)}) = (\vec{u}^{(m)} - \vec{u}^{*})$ where $\vec{u}^{*}$ is the fixed point of the Jacobi iteration. Then for $\nu > 0$

$$e(\vec{u}^{(\nu)}) = e((D^{-1}T)^{\nu}\vec{u}^{(0)}) = (D^{-1}T)^{\nu}e(\vec{u}^{(0)}) \tag{4}$$

which converges when $\rho(D^{-1}T) < 1$ since $\rho((D^{-1}T)^{\nu}) = (\rho(D^{-1}T))^{\nu}$. In order for the algorithm to correctly reduce the error it is necessary to compute the desired load at each time step to an appropriate accuracy. In order to quantify the error define the infinity norm

$$||e(\vec{u}^{(m)})||_{\infty} = \max_{x,y,z}\left|e(u^{(m)}_{x,y,z})\right| = \max_{x,y,z}\left|u^{(m)}_{x,y,z} - u^{*}_{x,y,z}\right|$$

Using this norm define a necessary condition to improve the accuracy of the solution $\vec{u}$ by a factor $\alpha$ in $\nu$ steps to be $||e(\vec{u}^{(\nu)})||_{\infty} \leq \alpha||e(\vec{u}^{(0)})||_{\infty}$. From (4) this is satisfied when $(\rho(D^{-1}T))^{\nu} \leq \alpha$ and thus (1)

$$\nu = \left\lceil \frac{\ln\alpha}{\ln\rho(D^{-1}T)} \right\rceil = \left\lceil \frac{\ln\alpha}{\ln\frac{6\alpha}{1+6\alpha}} \right\rceil \tag{5}$$

The method is unconditionally stable and the cost of this stability is small (in fact it is free for $0.833 \leq \alpha < 1$). This suggests the possible use of large time steps to deal with worst case disturbances.

## ELAPSED TIME FOR THE DIFFUSION

This section determines the number of artificial time steps $\tau$ required to reduce the load imbalance by a factor $\alpha$. It does this by considering the eigenstructure of the finite difference equation (22) which is rearranged to express the change in load with each iteration

$$\begin{aligned} u_{x,y,z}(t+dt) - u_{x,y,z}(t) = \alpha\,[u_{x+1,y,z}(t+dt) + u_{x-1,y,z}(t+dt) \\ +u_{x,y+1,z}(t+dt) + u_{x,y-1,z}(t+dt) \\ +u_{x,y,z+1}(t+dt) + u_{x,y,z-1}(t+dt) \\ -6u_{x,y,z}(t+dt)] \end{aligned}$$

or as a vector equation with matrix operator L

$$\vec{u}(t+dt) - \vec{u}(t) = \alpha \mathrm{L}\vec{u}(t+dt) \tag{6}$$

Any load distribution $\vec{u}(t)$ can be written as a weighted superposition of eigenvectors $\vec{x}$ of L

$$\vec{u}(t) = \sum_{i,j,k} a_{i,j,k}(t)\vec{x}_{i,j,k}$$

Using this fact rewrite the vector equation (6) as

$$\sum_{i,j,k} a_{i,j,k}(t+dt)\vec{x}_{i,j,k} - \sum_{i,j,k} a_{i,j,k}(t)\vec{x}_{i,j,k} = \alpha \sum_{i,j,k} \mathrm{L}a_{i,j,k}(t+dt)\vec{x}_{i,j,k} \tag{7}$$

Using the definition of $\mathrm{L}\vec{x}_{i,j,k}$ and the eigenvalues of L

$$\mathrm{L}\vec{x}_{i,j,k} = -\lambda_{i,j,k}\vec{x}_{i,j,k}$$

$$\lambda_{i,j,k} = 2\left[3 - \cos\left(2\pi i/n^{1/3}\right) - \cos\left(2\pi j/n^{1/3}\right) - \cos\left(2\pi k/n^{1/3}\right)\right] \tag{8}$$

(7) can be further simplified to

$$\sum_{i,j,k} \left(a_{i,j,k}(t+dt)\vec{x}_{i,j,k}\left[1+\alpha\lambda_{i,j,k}\right] - a_{i,j,k}(t)\vec{x}_{i,j,k}\right) = 0$$

and by the completeness and orthonormality of the eigenvectors

$$a_{i,j,k}(t+dt)\left[1+\alpha\lambda_{i,j,k}\right] - a_{i,j,k}(t) = 0$$

$$a_{i,j,k}(dt) = \frac{a_{i,j,k}(0)}{1+\alpha\lambda_{i,j,k}} \tag{9}$$

It is apparent from equation (9) that convergence of the individual components is dependent upon the eigenvalues $\lambda_{i,j,k}$. Reducing the amplitude of an arbitrary component $i,j,k$ by $\alpha$ in $\tau$ steps of the method requires $[1+\alpha\lambda_{i,j,k}]^{-\tau} \leq \alpha$. The worst case occurs for the smallest positive eigenvalue $\lambda_{0,0,1} = (2-2\cos(2\pi/n^{1/3}))$ which corresponds to a smooth sinusoidal disturbance with period equal to the length of the computational grid. To reduce such a disturbance requires

$$\tau = \left\lceil \frac{\ln \alpha^{-1}}{\ln\left[1+\alpha\left(2-2\cos\frac{2\pi}{n^{1/3}}\right)\right]} \right\rceil \tag{10}$$

Convergence of this slowest component approaches $\ln \alpha^{-1}$ for large $n$ since

$$\lim_{n\to\infty} \ln\left[1+\alpha\left(2-2\cos\frac{2\pi}{n^{1/3}}\right)\right] = 1$$

Convergence of highest wavenumber component $\lambda_{(n^{1/3})/2-1,(n^{1/3})/2-1,(n^{1/3})/2-1}$ is rapid because

$$\tau = \left\lceil \frac{\ln \alpha^{-1}}{\ln\left[1+(6-\epsilon)\alpha\right]} \right\rceil \tag{11}$$

## ANALYSIS OF A POINT DISTURBANCE

This section considers a specific case of a point disturbance and derives an inequality relating $\tau$, $n$ and $\alpha$. The purpose in doing this is to provide an exact prediction of convergence of a known case in order to demonstrate scaling properties. The procedure followed is to describe the initial amplitudes of the components of the disturbance and then solve an inequality which describes the magnitude of the disturbance over time. A periodic domain is assumed for the purpose of analysis. Simulations presented later in this paper verify that convergence is similar on aperiodic domains.

The following text uses the Poisson bracket $\langle\cdot,\cdot\rangle$ to represent the inner product operator. When discussing loads or eigenvectors it uses $\vec{u}[x,y,z]$ or $\vec{x}_{i,j,k}[x,y,z]$ to denote the vector element which corresponds to location $x,y,z$ of the computational grid with the convention that $[0,0,0]$ is the first element of the vector. Then the initial disturbance confined to a particular processor $x,y,z$ can be written as a superposition of eigenvectors of L

$$\vec{u}(0) = \sum_{l,m,n} a_{l,m,n}(0)\vec{x}_{l,m,n} \tag{12}$$

Assume the initial disturbance $u(0)$ to be zero at every element except $[x, y, z]$. Then

$$\langle \vec{x}_{i,j,k}, \vec{u}(0) \rangle = \vec{x}_{i,j,k}[x, y, z] \tag{13}$$

This is equal to the initial amplitude $a_{i,j,k}(0)$ of each eigenvector $\vec{x}_{i,j,k}$

$$\begin{aligned} \langle \vec{x}_{i,j,k}, \vec{u}(0) \rangle &= \left\langle \vec{x}_{i,j,k}, \sum_{l,m,n} a_{l,m,n}(0) \vec{x}_{l,m,n} \right\rangle \\ &= \sum_{l,m,n} \langle \vec{x}_{i,j,k}, \vec{x}_{l,m,n} \rangle \, a_{l,m,n}(0) \\ &= \sum_{l,m,n} a_{l,m,n}(0) \delta_{il} \delta_{jm} \delta_{kn} \\ &= a_{i,j,k}(0) \end{aligned} \tag{14}$$

The computational domain has periodic boundary conditions and as a result the origin of the coordinate system is arbitrary. Without loss of generality place the origin at the source of the disturbance and take $x = y = z = 0$. This has no effect on the eigenvectors $\vec{x}_{i,j,k}$ and from (13), (14)

$$a_{i,j,k}(0) = \vec{x}_{i,j,k}[0,0,0] \tag{15}$$

Placing the origin at the source of the disturbance is particularly convenient when we consider the first element of the eigenvectors $\vec{x}_{i,j,k}[0,0,0]$. L has $\left(n^{1/3}\right)/2$ distinct eigenvalues $\lambda_{i,j,k}$ each of algebraic multiplicity two. Each of these eigenvalues has geometric multiplicity eight, ie. has eight linearly independent associated eigenvectors of unit length

$$\vec{x}_{i,j,k}[x, y, z] = c_{i,j,k} F_1 \left(2\pi x i / n^{1/3}\right) F_2 \left(2\pi y j / n^{1/3}\right) F_3 \left(2\pi z k / n^{1/3}\right) \tag{16}$$

where each $F_i$ is either sin or cos. By choosing $x = y = z = 0$ this expression (16) is zero except for the single eigenvector for which $F_1(x) = F_2(x) = F_3(x) = \cos(x)$. Without loss of generality restrict further consideration to initial disturbances of the form

$$u[0,0,0](0) = \sum_{i,j,k} c_{i,j,k} \vec{x}_{i,j,k}[0,0,0] = \sum_{i,j,k} c_{i,j,k}^2 \tag{17}$$

From (9) define the time dependent disturbance at any location $x', y', z'$

$$\vec{u}[x', y', z'](\tau dt) = \sum_{i,j,k} c_{i,j,k} \left[1 + \alpha \lambda_{i,j,k}\right]^{-\tau} \vec{x}_{i,j,k}[x', y', z']$$

| $\tau(\alpha, n)$ |  | $n$ (total processors) |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
|  |  | 64 | 512 | 4,096 | 8,000 | 32K | 256K | $10^6$ |
|  | 0.1 | 7 | 6 | 6 | 5 | 5 | 5 | 5 |
| $\alpha$ | 0.01 | 152 | 213 | 229 | 173 | 157 | 145 | 141 |
|  | 0.001 | 2,749 | 5,763 | 10,031 | 10,139 | 9,082 | 7,564 | 7,003 |

Table 1: Solutions to equation (20) for increasing processor count $n$ and accuracy $\alpha$. $\tau$ represents the number of exchange steps in the method. See figure 1.

$$= \sum_{i,j,k} c_{i,j,k}^2 \left[1 + \alpha\lambda_{i,j,k}\right]^{-\tau} \cos\left(2\pi x' i / n^{1/3}\right) \cos\left(2\pi y' j / n^{1/3}\right) \cos\left(2\pi z' k / n^{1/3}\right) \tag{18}$$

The appendix demonstrates that $c_{i,j,k} = (8/n)^{1/2}$ and thus the disturbance is a summation of equally weighted eigenvectors. From (17) and (18) the time dependent disturbance at $0, 0, 0$ is therefore

$$\vec{u}[0,0,0](\tau dt) = \frac{8}{n} \sum_{i,j,k} \left[1 + \alpha 2 \left(3 - \cos\frac{2\pi i}{n^{1/3}} - \cos\frac{2\pi j}{n^{1/3}} - \cos\frac{2\pi k}{n^{1/3}}\right)\right]^{-\tau} \tag{19}$$

Solving $\vec{u}[0,0,0](\tau dt) \leq \alpha$ yields

$$\frac{8}{n} \sum_{i,j,k} \left[1 + \alpha 2 \left(3 - \cos\frac{2\pi i}{n^{1/3}} - \cos\frac{2\pi j}{n^{1/3}} - \cos\frac{2\pi k}{n^{1/3}}\right)\right]^{-\tau} \leq \alpha \tag{20}$$

where $i, j, k$ are indexed from 0 to $\left(n^{1/3}\right)/2 - 1$ and the case $i = j = k = 0$ is omitted.

Table 1 and figure 1 present solutions of the inequality (20). These are exact predictions of the number $\tau$ of exchange steps which must occur to reduce a point disturbance by a factor $\alpha$ on a periodic domain. Figures 2 and 3 are simulated results for two CFD cases. The first case of partitioning an unstructured grid is a point disturbance. In the simulation $\tau$ is observed to match the theoretical prediction exactly. The second case of rebalancing after a grid adaptation demonstrates the value of estimating $\tau$ from simulations. The initial disturbance is not a point and the simulation is observed to require 170 exchange steps before the worst case discrepancy drops to 10% of it's original value.

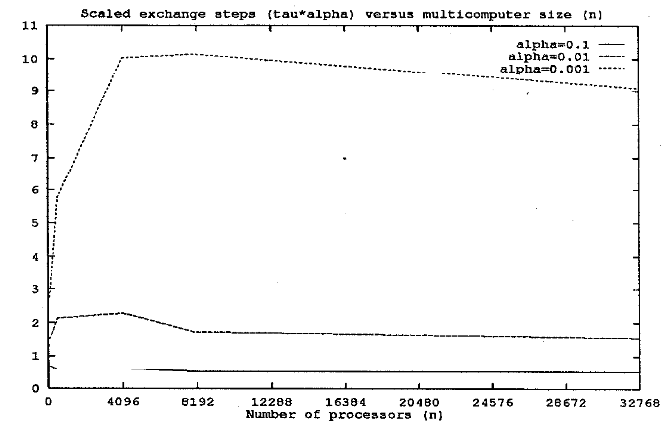

Figure 1: Scaled number of exchange steps $\tau\alpha$ to achieve accuracy $\alpha$ for various $n, \alpha$ following a point disturbance. See table 1 and equation (20). Each line is scaled by $\alpha$. All lines are initially increasing for small $n$ and asymptotically decreasing for larger $n$ demonstrating weak superlinear speedup.

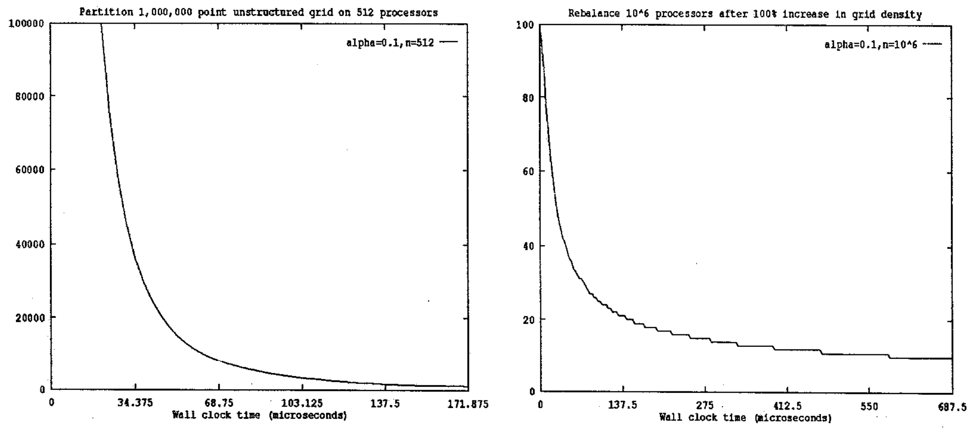

Figure 2: Time course of disturbances for simulated CFD cases. Left: largest discrepancy among 512 processors partitioning an unstructured computational grid. The initial disturbance of 1,000,000 points confined to a single processor is reduced by 90% after 6 exchanges (20.625 $\mu s$) in exact agreement with theory ( $\tau(0.1, 512)$ in table 1). Right: largest discrepancy among 1,000,000 processors rebalancing a disturbance following a bow shock adaptation. $\alpha = 0.1, \nu = 3$ in both cases. Wall clock times assume a 32 MHz J-machine. The interval between exchange steps is 3.4375 $\mu s$.

## 5 Simulations

This section presents simulations of three cases in which the method is useful. The first case is a bow shock resulting from a million grid point CFD calculation [23]. The disturbance dissipates rapidly and is nearly gone after 10 exchange steps. The second case computes an initial load distribution for a million point unstructured grid problem on a 512 node multicomputer. The simulation suggests the method may be highly competitive with Lanczos based approaches presented recently in [3, 20]. The third case simulates a multicomputer operating system under conditions of random load injection. This case demonstrates that the method can effectively balance large disturbances which occur frequently and randomly.

Parameters for the simulations are based on two scalable multicomputers. The first is a 512 node J-machine [19] which is in use at Caltech for research in CFD and scalable concurrent computing. The second is a hypothetical 1,000,000 node J-machine. These two design points represent a continuum of scalable multicomputers. The method is practical at both ends of this continuum and presumably at all points in between.

Wall clock times are based on a hand coded implementation of the method in J-machine assembler and assumes 32 MHz processors. Each repetition of the method requires 110 instruction cycles in 3.4375 $\mu s$. All simulations are run with $\alpha = 0.1$ and $\nu = 3$ resulting in 3 iterations between each exchange of work.

### 5.1 Bow Shock Adaptation

In CFD calculations it is common to adapt a computational grid in response to properties of a developing solution. This simulation considers an adaptation that results from a bow shock in front of a Titan IV launch vehicle with two boosters (figure 3). The grid has been adapted by doubling the density of points in each area of the bow shock. As a result the initial disturbance shows locations in the multicomputer where the workload has increased by 100% due to the introduction of new points. After 10 exchanges of work the imbalance has already decreased dramatically. After 70 exchanges the disturbance is scarcely visible. This simulation assumes a 1,000,000 processor J-machine. This example illustrates the weak persistence of low spatial frequencies.

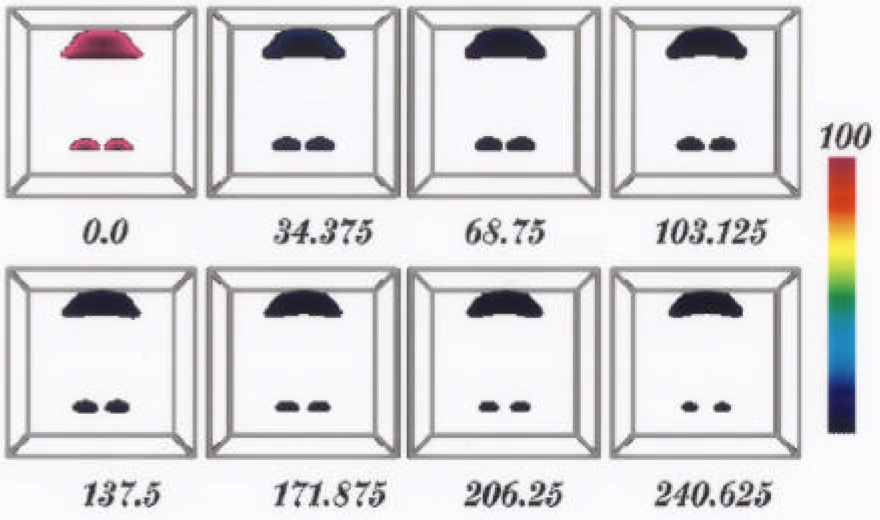

Figure 3: Disturbance following a bow shock adaptation of a computational grid on a million processor J-machine. First frame is the initial disturbance resulting from the adaptation. Subsequent frames are separated by 10 exchange steps. The disturbance is reduced dramatically by the second frame. After 70 exchange steps only weak low frequency components remain.

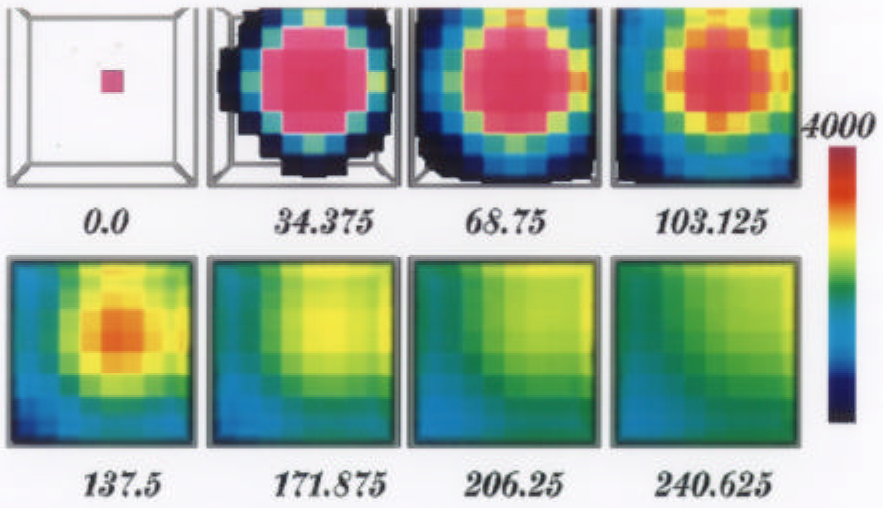

Figure 4: Disturbance during an initial load of a million point unstructured computational grid onto a 512 processor J-machine. The first frame represents the entire grid assigned to a host node on the multicomputer. This is a point disturbance and the resulting behavior is in exact agreement with the analysis presented earlier in this paper. Subsequent frames are separated by 10 exchange steps. After 70 exchange steps the workload is already roughly balanced. A balance within 1 grid point was achieved after 500 exchange steps.

## 5.2 Partitioning an Unstructured Grid

In parallel CFD applications the static load balancing problem has been the subject of recent attention [3, 20]. In addition to finding an equitable distribution of work this problem must observe the additional constraint of preserving adjacency relationships among elements of an unstructured computational grid. The load balancing method presented in this paper can satisfy the adjacency constraint if at each exchange step it selects exchange candidates that observe the constraint.

This simulation models an initial point disturbance by assigning 1,000,000 points of an unstructured computational grid to a single host node of a 512 processor J-machine (see figure 4). It executes the algorithm while observ-

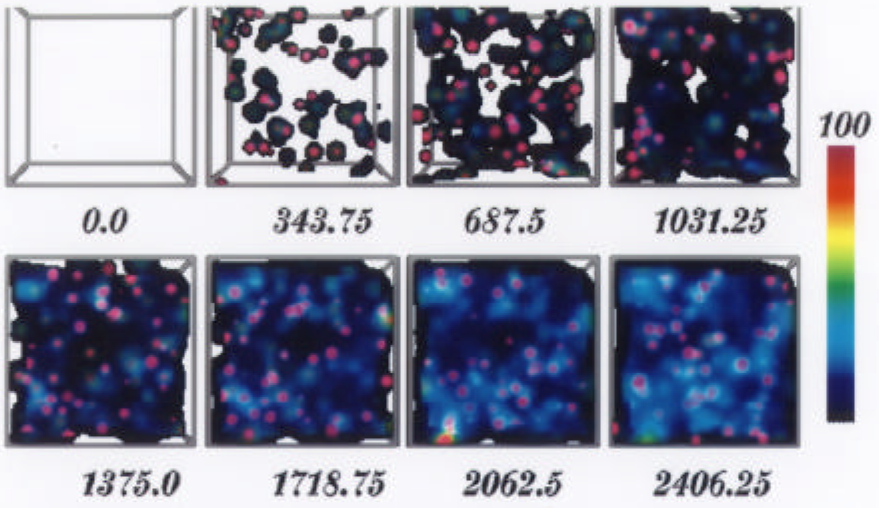

Figure 5: Disturbances resulting from rapid injection of large random loads on a million processor J-machine. After each exchange step a point disturbance is introduced at a randomly chosen processor. The average value of each point disturbance is 30,000 times the initial system load average. The interval between successive frames is 100 exchange steps. After 700 injections the worst case discrepancy was 15,737 times the initial load average. This demonstrates the algorithm was balancing the load faster than disturbances were created. After load injection ceased an additional 100 repetitions with no new disturbance reduced the worst case discrepancy from 15,737 to 50 times the initial load average.

ing the adjacency constraint at each exchange step. The initial disturbance was reduced by 90% after 6 exchange steps in exact agreement with theory ($\tau(0.1, 512)$ in table 1). After 59 exchange steps the worst case discrepancy was 999 grid points. After 162 steps the worst case discrepancy was 200 grid points, 10% of the load average. A balance within 1 grid point was achieved on the 500th step.

## 5.3 Random Load Injection

To be useful in practical contexts the method must be able to rebalance disturbances faster than they arise. This simulation (figure 5) demonstrates the behavior of the method on a 1,000,000 processor J-machine under demanding conditions. An initially balanced distribution is disrupted repeatedly by large injections of work at random locations. Injection magnitudes are uniformly distributed between 0 and 60,000 times the initial load average. The simulation alternates repetitions of the algorithm with injections at randomly chosen locations. After 700 repetitions and injections the worst case discrepancy was 15,737 times the initial load average. This is less than the average injection magnitude of 30,000 at each repetition. This demonstrates the method was balancing the load faster than the injections were disrupting it. The last random injection occured on step 700. After 100 additional exchange steps without intervening injections the worst case discrepancy had reduced from 15,737 to 50 times the initial load average.

## 6 Summary and Discussion

This paper has demonstrated that a diffusion based load balancing method is efficient, reliable, and scalable. It has shown through rigorous theory and empirical simulation that the method is inexpensive for important generic problems in CFD. This section discusses a few remaining points and indicates future directions for this research.

An important property of the method is it's ability to preserve adjacency relationships among elements of a computational domain. Preserving adjacency permits CFD calculations to minimize their communication costs. The method preserves adjacency if it does so at each exchange step.

To make this requirement concrete consider an unstructured computational grid for a CFD calculation. When the time comes for the load balancing method to select grid points to exchange with neighboring processors it selects points in such a way that average pairwise distance among all points is minimal. One way to do this is to assume that each processor represents a volume of the computational domain and to select for exchange those grid points which occupy the exterior of the volume. The selected points would transfer to adjacent volumes where their neighbors in the computational grid already reside. In order for this to be practical it must be inexpensive to identify these exterior points. In problems where the computational grid has been generated contiguously this is presumably a simple matter because adjacent points will have data structures that identify their neighbors. In more general problems where the data is not already ordered the use of priority queues appears promising due to their $O(n \log n)$ complexity.

It is worth noting that the method can be used to rebalance a local portion of a computational domain without interrupting the computation which is occurring on the rest of the domain. This can be useful in CFD problems where some portions of the domain converge more quickly than others and adaptation might occur locally and frequently.

This paper presented the method and analysis for a domain which has periodic boundary conditions and is logically spherical. In practice multicomputer meshes are rarely periodic. In our simulations we implemented aperiodic boundaries by imposing the Neumann condition $\partial u/\partial x = 0$ in each direction $x, y, z$. This requires a simple modification to the iteration (2) so that processors immediately outside the mesh appear to have the same workload as processors one step inside the mesh. For example, if the processors are indexed from 1 through $n$ in the $x$ dimension then $u_{0,y,z} = u_{2,y,z}$ and $u_{n+1,y,z} = u_{n-1,y,z}$.

The algorithm is presented for three dimensional scalable multicomputers. It reduces for two dimensional cases by redefining $\nu$ and the iteration (2) as follows:

$$\nu = \left\lceil \frac{\ln \alpha}{\ln \frac{4\alpha}{1+4\alpha}} \right\rceil \geq 1$$

$$u_{x,y}^{(m)} = \frac{u_{x,y}^{(0)}}{1+4\alpha} + \left(\frac{\alpha}{1+4\alpha}\right)\left(u_{x+1,y}^{(m-1)} + u_{x-1,y}^{(m-1)} + u_{x,y+1}^{(m-1)} + u_{x,y-1}^{(m-1)}\right)$$

The worst case behavior of the method is determined by disturbances of low spatial frequency. This is the basis of the objections in [11] and a conventional response would be to apply a multigrid method [11, 14]. Such a method can have logarithmic scaling in $n$ and $O(n)$ convergence. The analysis presented in this paper demonstrates that wall clock times can actually decrease as $n$ increases and this suggests considering other methods of treating the worst case disturbance. One such method would be to use very large time steps in order to accelerate convergence of the low frequency components. The unconditional stability of this method makes this an attractive option. Although this would increase the error in the high frequency components these components can be quickly corrected by local iterations. We are presently considering the costs associated with such iterations.

## 7 Appendices

This appendix presents technical justifications for statements relied upon in the analysis.

FINITE DIFFERENCE FORMULATION OF THE HEAT EQUATION

Consider the parabolic heat equation in three dimensions

$$u_t = \nabla^2 u = u_{xx} + u_{yy} + u_{zz} \tag{21}$$

Taylor expanding in $t$ with all derivatives evaluated at $(x, y, z, t)$

$$\begin{aligned} u(x,y,z,t+dt) &= u(x,y,z,t) + u_t dt + O(dt^2) \\ u_t &= \left(\frac{u(x,y,z,t+dt) - u(x,y,z,t)}{dt}\right) + O(dt) \end{aligned}$$

Obtain the second order terms by expanding in spatial variables where omitted coordinates are interpreted as $(x, y, z, t)$

$$\begin{aligned} u(x+dx,\cdot,\cdot,\cdot) &= u(x,\cdot,\cdot,\cdot) + u_x dx + \\ & \quad u_{xx}\frac{dx^2}{2} + u_{xxx}\frac{dx^3}{6} + O(dx^4) \\ u(x-dx,\cdot,\cdot,\cdot) &= u(x,\cdot,\cdot,\cdot) - u_x dx + \\ & \quad u_{xx}\frac{dx^2}{2} - u_{xxx}\frac{dx^3}{6} + O(dx^4) \\ u(x+dx,\cdot,\cdot,\cdot) + u(x-dx,\cdot,\cdot,\cdot) &= 2u(x,\cdot,\cdot,\cdot) + u_{xx}dx^2 + O(dx^4) \end{aligned}$$

$$u_{xx} = \left(\frac{u(x+dx,\cdot,\cdot,\cdot) + u(x-dx,\cdot,\cdot,\cdot) - 2u(x,\cdot,\cdot,\cdot)}{dx^2}\right) + O(dx^2)$$

Similar expansions in $y, z$ show that the heat equation can be rewritten

$$\frac{u(\cdot,\cdot,\cdot,t+dt) - u(\cdot,\cdot,\cdot,t)}{dt} = \left(\frac{u(x+dx,\cdot,\cdot,\cdot) + u(x-dx,\cdot,\cdot,\cdot) - 2u(\cdot,\cdot,\cdot,\cdot)}{dx^2}\right)$$

$$+ \left(\frac{u(\cdot,y+dy,\cdot,\cdot) + u(\cdot,y-dy,\cdot,\cdot) - 2u(\cdot,\cdot,\cdot,\cdot)}{dy^2}\right)$$

$$+\left(\frac{u(\cdot,\cdot,z+dz,\cdot)+u(\cdot,\cdot,z-dz,\cdot)-2u(\cdot,\cdot,\cdot,\cdot)}{dz^2}\right)+O(dt,dx^2,dy^2,dz^2)$$

From (21) the finite difference formulation is

$$\frac{u(x,y,z,t+dt)-u(x,y,z,t)}{dt}=\frac{1}{dx^2}(u(x+dx,y,z,t)+u(x-dx,y,z,t)+$$
$$u(x,y+dy,z,t)+u(x,y-dy,z,t)+u(x,y,z+dz,t)+$$
$$u(x,y,z-dz,t)-6u(x,y,z,t))$$

Setting $\alpha=\frac{dt}{dx^2}$ and taking the spatial terms on the right at time $t+dt$ yields an unconditionally stable implicit scheme

$$u(x,y,z,t)=(1+6\alpha)u(x,y,z,t+dt)-\alpha\left[u(x+dx,y,z,t+dt)+\right. \tag{22}$$
$$u(x-dx,y,z,t+dt)+u(x,y+dy,z,t+dt)+u(x,y-dy,z,t+dt)+$$
$$\left.u(x,y,z+dz,t+dt)+u(x,y,z-dz,t+dt)\right]$$

JACOBI ITERATION TO COMPUTE $A^{-1}u$

In order to compute solutions at successive time intervals $dt$ we must invert the relationship $u^{(t)}=Au^{(t+dt)}$ by solving

$$u^{(t+dt)}=A^{-1}u^{(t)} \tag{23}$$

From (22) it is apparent that $A$ has diagonal terms $(1+6\alpha)$ and six offdiagonals $\alpha$. Let $A=(D-T)$ where D is this diagonal. Then $(D-T)u^{(t+dt)}=u^{(t)}$ implies $u^{(t+dt)}=D^{-1}Tu^{(t+dt)}+D^{-1}u^{(t)}$. This relation is satisfied by fixed points of the Jacobi iteration

$$\left[u^{(t+dt)}\right]^{(m)}=D^{-1}T\left[u^{(t+dt)}\right]^{(m-1)}+D^{-1}u^{(t)} \tag{24}$$

The matrix $D^{-1}T$ has a zero diagonal and six offdiagonal terms $\frac{\alpha}{1+6\alpha}$. $D^{-1}$ is a diagonal matrix with terms $\frac{1}{1+6\alpha}$. The iteration (24) is the central loop of the method (2). Note that this iteration is everywhere convergent with spectral radius defined by (3).

UNIT IMPULSE DERIVATION

This section demonstrates that the eigenvector normalization constant $c_{i,j,k}$ is equal to $(8/n)^{1/2}$ for all eigenvectors $\vec{x}_{i,j,k}$. From (16) a necessary condition for a normalized eigenvector is

$$\begin{aligned}
1 &= c_{i,j,k}^2 \sum_{x,y,z} \cos^2\left(2\pi\frac{xi}{n^{1/3}}\right)\cos^2\left(2\pi\frac{yj}{n^{1/3}}\right)\cos^2\left(2\pi\frac{zk}{n^{1/3}}\right) \\
&= c_{i,j,k}^2 \frac{1}{8}\sum_{x,y,z}\left(1+\cos 4\pi\frac{xi}{n^{1/3}}\right)\left(1+\cos 4\pi\frac{yj}{n^{1/3}}\right)\left(1+\cos 4\pi\frac{zk}{n^{1/3}}\right) \\
&= c_{i,j,k}^2 \frac{1}{8}\sum_{x,y,z}\left\{\left(1+\cos 4\pi\frac{yj}{n^{1/3}}\right)\left(1+\cos 4\pi\frac{zk}{n^{1/3}}\right)\right. \\
&\quad \left. +\sum_x \cos\left(4\pi\frac{xi}{n^{1/3}}\right)\sum_{y,z}\cos\left(4\pi\frac{yj}{n^{1/3}}\right)\left(1+\cos 4\pi\frac{zk}{n^{1/3}}\right)\right\} \qquad (25)
\end{aligned}$$

Simplify the preceding expression by the following

**Lemma 1**

$$\sum_x \cos\left(4\pi\frac{xi}{n^{1/3}}\right) = 0$$

*Proof:*

$$\begin{aligned}
\sum_x \cos\left(4\pi\frac{xi}{n^{1/3}}\right) &= \sum_x Re\left(e^{\imath\frac{4\pi x i}{n^{1/3}}}\right) \\
&= Re\sum_x e^{\imath\frac{4\pi x i}{n^{1/3}}} \\
&= Re\sum_x \left(e^{\imath\frac{4\pi i}{n^{1/3}}}\right)^x \\
&= Re\left[\frac{e^{\imath\frac{4\pi i}{n^{1/3}}}\left(1-\left(e^{\imath\frac{4\pi i}{n^{1/3}}}\right)^{n^{1/3}}\right)}{1-e^{\imath\frac{4\pi i}{n^{1/3}}}}\right] \\
&= 0
\end{aligned}$$

*Q.E.D.*

Repeated application of lemma (1) to equation (25) yields

$$\begin{aligned}
1 &= c_{i,j,k}^2 \frac{1}{8} \sum_{x,y,z} \left(1 + \cos 4\pi \frac{yj}{n^{1/3}}\right)\left(1 + \cos 4\pi \frac{zk}{n^{1/3}}\right) \\
&= c_{i,j,k}^2 \frac{1}{8} \left[\sum_{x,y,z} \left(1 + \cos 4\pi \frac{zk}{n^{1/3}}\right) + \sum_{x,y,z} \left(\cos 4\pi \frac{jy}{n^{1/3}}\right)\left(1 + \cos 4\pi \frac{zk}{n^{1/3}}\right)\right] \\
&= c_{i,j,k}^2 \frac{1}{8} \left[\sum_{x,y,z} 1 + \sum_{x,y,z} \left(\cos 4\pi \frac{zk}{n^{1/3}}\right)\right] \\
&= c_{i,j,k}^2 \frac{1}{8} n \qquad (26)
\end{aligned}$$

From which we conclude

$$c_{i,j,k} = \left(\frac{8}{n}\right)^{1/2} \qquad \forall\, i,j,k$$

## 8 Acknowledgements

We thank Michael Noakes for a concurrent implementation of the iteration (2) in J-Machine assembler. We are grateful to Andrew Conley for helpful discussions and criticisms.

## References

[1] Amdahl, G. Validity of the single processor approach to achieving large scale computing capabilities. *Proc. AFIPS Comput. Conf.* **30** (1967) 483–485.

[2] Athas, W. C. & Seitz, C. L. Multicomputers: message passing concurrent computers. *IEEE Comp.* **21** (1988) 9–24.

[3] Barnard, S. T. & Simon, H. D. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. To appear in *Concurrency: Pract. Exp.* (1993).

[4] Boillat, J. E. Load balancing and poisson equation in a graph. *Concurrency: Pract. Exp.* **2** (1990) 289–313.

[5] Brugé, F. & Fornili, S. L. A distributed dynamic load balancer and it's implementation on multi-transputer systems for molecular dynamics simulation. *Comp. Phys. Comm.* **60** (1990) 39–45.

[6] Cybenko, G. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.* **7** (1989) 279–301.

[7] Horn, R. A. & Johnson, C. R. *Matrix Analysis.* Cambridge University Press, New York, NY, 1991.

[8] Golub, G. H. & Van Loan, C. F. *Matrix Computations* (2nd edition). The Johns Hopkins University Press, Baltimore, MD, 1989.

[9] Gustafson, J. L. Reevaluating Amdahl's law. *Comm. ACM* **31** (1988) 532–533.

[10] Hofstee, H. P., Lukkien, J. J. & van de Snepscheut, J. L. A. A distributed implementation of a task pool. *Research Directions in High Level Parallel Programming Languages,* Banatre, J. P. & Le Metayer, D. (eds.), Lecture Notes in Computer Science **574** (1992) 338-348, Springer: New York.

[11] Horton, G. A multi-level diffusion method for dynamic load balancing. *Parallel Comp.* **19** (1993) 209–218.

[12] Kumar, V., Ananth, G. Y. & Rao, V. N. Scalable load balancing techniques for parallel computers. Preprint 92-021. Army High Performance Computing Research Center, Minneapolis, MN, 1992.

[13] Lin, F. C. H. & Keller, R. M. The gradient model load balancing method. *IEEE Trans. Soft. Eng.* **SE-13** (1987) 32–38.

[14] McCormick, S. F., *Multigrid Methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[15] Marinescu, D. C. & Rice, J. R. Synchronization and load imbalance effects in distributed memory multi-processor systems. *Concurrency: Pract. Exp.* **3** (1991) 593–625.

[16] Keckler, S. W. & Dally, W. J. Processor coupling: integrating compile time and run time scheduling for parallelism. *Proc. ACM 19th Int'l. Symp. on Comp. Arch.*, Queensland, Australia (1992) 202-213.

[17] Ni, L. M., Xu, C. & Gendreau, T. B. A distributed drafting algorithm for load balancing. *IEEE Trans. Soft. Eng.* **SE-11** (1985).

[18] Nicol, D. M. & Saltz, J. H. Dynamic remapping of parallel computations with varying resource demands. *IEEE Trans. Comp.* **37** (1988).

[19] Noakes, M. & Dally, W. J. System design of the J-machine. *Proc. 6th MIT Conf. on Advanced Research in VLSI.* Dally, W. J. (Ed.). MIT Press, Cambridge, MA, 1990, pp. 179–194.

[20] Pothen, A., Simon, H. D. & Liou, K. Partitioning Sparse Matrices with Eigenvectors of Graphs. *SIAM J. Matrix Anal.* **11** (1990) 430–452.

[21] Seitz, C. L. Mosaic C: an experimental fine-grain multicomputer. *Proc. International Conference Celebrating the 25th Anniversary of INRIA, Paris, France, December 1992*, Springer-Verlag, New York, NY, 1992.

[22] Shen, S. Cooperative Distributed Dynamic Load Balancing. *Acta Informatica* **25** (1988) 663–676.

[23] Wang, J. C. T. & Taylor, S. A Concurrent Navier-Stokes Solver for Implicit Multibody Calculations. To appear in *Proc. Parallel-CFD '93*, Paris, France, June 1993.

[24] Rowell, J. Lessons Learned on the Delta. *High Perf. Comput. Rev.* **1** (1993) 21–24.